M 1 101 – Systèmes d'exploitation – Architecture des microprocesseurs

S1, année 2020 – 2021

Département Informatique

IUT de Bordeaux

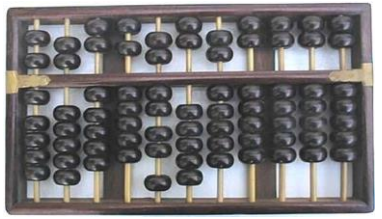
Contenu du cours M 1 101

1. Histoire de l'informatique
2. Codage de l'information
3. Architecture des microprocesseurs
    1. Unité centrale du microprocesseur
    2. Notion de programmation bas niveau
    3. Mémoire cache

1. Histoire de l'Informatique

-**3000** : Période de l'empereur Chinois **Fou-Hi** dont le symbole magique, l'octogone à trigramme contient les 8 premiers nombres représentés sous forme binaire par des traits interrompus ou non : 000 001 010 011 etc...

-**500** : Moyen Orient : le boulier



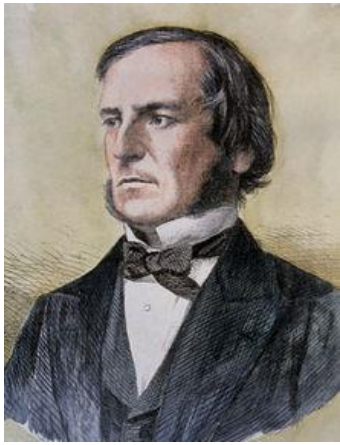
Mécanisation du calcul Schickard (1623), Pascal (1642), Leibniz (1673) → réalisation des additions, soustractions, multiplications et mémorisation des résultats intermédiaires grâce à des systèmes mécaniques tels que des roues dentées

Automatisation du travail Falcon (1728), Jacquard (1805) métier à tisser utilisant des cartes perforées

Calcul automatique Babbage (1833) **Babbage** imagine et tente de réaliser une **machine à différences** puis une **machine analytique** qui contient les concepts de ce que sera l'ordinateur moderne : unité de calcul, mémoire, registre et entrée des données par carte perforée.

1. Histoire de l'Informatique

1854 : **Boole** publie un ouvrage dans lequel il démontre que tout processus logique peut être décomposé en une suite d'opérations logiques (ET, OU, NON) appliquées sur deux états (ZERO-UN, OUI-NON, VRAI-FAUX, OUVERT-FERME)



Hollerith (1884) crée une tabulatrice à cartes perforées

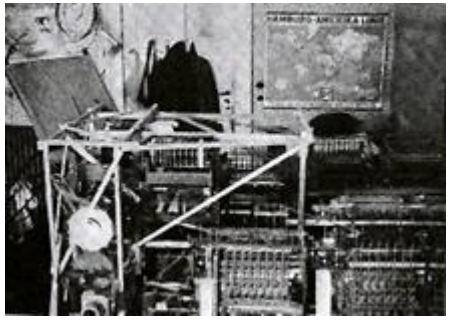
1904 : Invention du premier tube à vide, la **diode** par John Fleming

1924 : La firme créée par Hollerith en 1896, est renommée International Busines Machine

1935 : **IBM** commercialise l'**IBM 601**, un calculateur à relais utilisant des cartes perforées capable de réaliser une multiplication en une seconde

1. Histoire de l'Informatique

1937 : Alan Turing publie un document sur les nombres calculables (Machine de Turing)



1938 : Thèse de **Shannon** → le parallèle entre les circuits électriques et l'algèbre Booléenne. Il définit le chiffre binaire : **bit** (BInary digiT).

1938 : Création du premier ordinateur binaire programmable mais mécanique « **Versuchmodell 1 »** ou **Z1** par **Konrad Zuse**

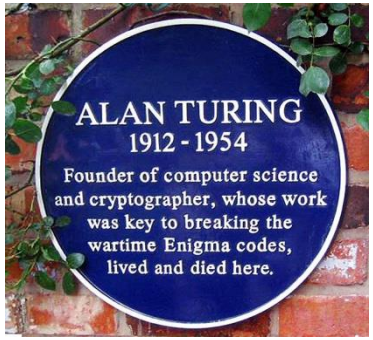
1939 : Réalisation d'un deuxième ordinateur, le **Z2** en remplaçant une partie des pièces mécaniques du **Z1** par des relais électromécaniques de téléphone (puis Z3 premier véritable ordinateur détruit en 1945 et Z4, Z1 et Z4 en photo)

1940 : les calculateurs **Robinson** et **Colossus** avec les concepts d'arithmétique binaire, d'horloge interne, de mémoire tampon, de lecteurs de bande, d'opérateurs booléens, de sous programmes et d'imprimantes.

1. Histoire de l'Informatique

1940 : Pour décrypter les messages de l'armée allemande, les Anglais mettent au point sur le site de **Bletchey Park**, les premières machines qui intègrent les concepts d'arithmétique binaire, d'horloge interne, de mémoire tampon, de lecteurs de bande, de sous-programmes et d'imprimantes. « Secret défense » jusqu'en 1975.

1945 : **Alan Turing** a joué un rôle majeur dans la victoire des Alliés (gain de deux ans de guerre), mais marginalisé par la société et réhabilité par la reine Élisabeth II en 2013.



De nombreux films et documentaires relatent cet épisode historique:

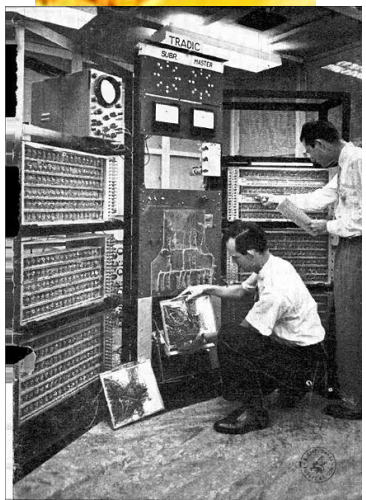
Imitation Game (2014), La drôle de guerre d'Alan Turing (2014), U571 (2000), John Von Neuman, prophète du XXI ème siècle (2015) …

1. Histoire de l'Informatique

1945 : **John Von Neuman**, ayant rejoint l'équipe travaillant sur l'**ENIAC**, publie le premier rapport décrivant ce que devrait être un ordinateur à programme enregistré → **architecture Von Neuman**.

1946 : Création de l'**ENIAC** (Electronic Numerical Integrator and Computer) par **P. Eckert** et **J. Mauchly**. Un calculateur composé de 19000 tubes pèse 30 tonnes, occupe une surface de 72 m2 et consomme 140 kilowatts. Horloge : 100 KHz. Vitesse : environ 330 multiplications par seconde.

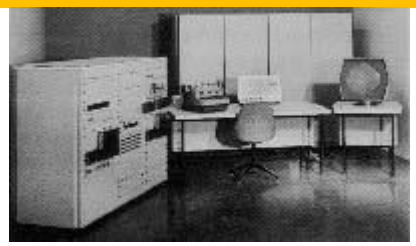
Décembre 1947 : Invention du **transistor** par **William Bradford Shockley**, **Walter H. Brattain** et **John Bardeen** dans les laboratoires de Bell Telephone.



1956 : Création du premier **ordinateur à transistors** par la Bell : le **TRADIC** qui amorce la seconde génération d'ordinateurs.

1957 : Création du premier langage de programmation universel, le **FORTRAN** (FORmula TRANslator) par **John Backus** d'**IBM**.

1. Histoire de l'Informatique



1959 : **Digital** crée le **PDP-1**, le premier mini ordinateur commercial interactif

Novembre 1971 : **Intel** commercialise le premier micro ordinateur **MCS-4** basé sur son tout nouveau microprocesseur 4004 et contenant aussi une Rom Intel 4001, une Ram Intel 4002 et un registre à décalage Intel 4003.

Années 90 accession des micro-ordinateurs au grand public, émergence du multimedia, d'internet, des jeux. Quasi monopole imposé et arrogant des PC (Personal Computer) et Microsoft.

Années 2000 introduction de l'ordinateur dans les activités quotidiennes de chacun, aide à la conduite automobile interactive, téléphonie portable, cuisine assistée, choix de programmes télévisés personnalisés…

DE NOS JOURS

Processeurs de plus en plus complexes : plusieurs coeurs, des lignes de caches, des coprocesseurs etc.

Évolution du nombre de transistors par processeur

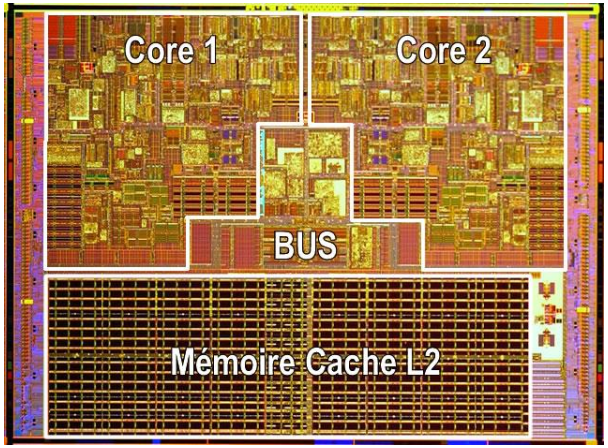
année	transistors	processeur
1971	2,300	Intel 4004, premier microprocesseur
1978	29,000	Intel 8086, premiers PC
1979	68,000	Motorola 68000
1989	1,180,000	Intel 80486
1993	3,100,000	Pentium
1997	9,500,000	Pentium III
2000	42,000,000	Pentium 4
2012	1,400,000,000	Quad-Core + GPU Core i7
2012	5,000,000,000	62-Core Xeon Phi
2014	5,560,000,000	18-core Xeon Haswell (photo)
2015	7,100,000,000	IBM z13 Storage Controller

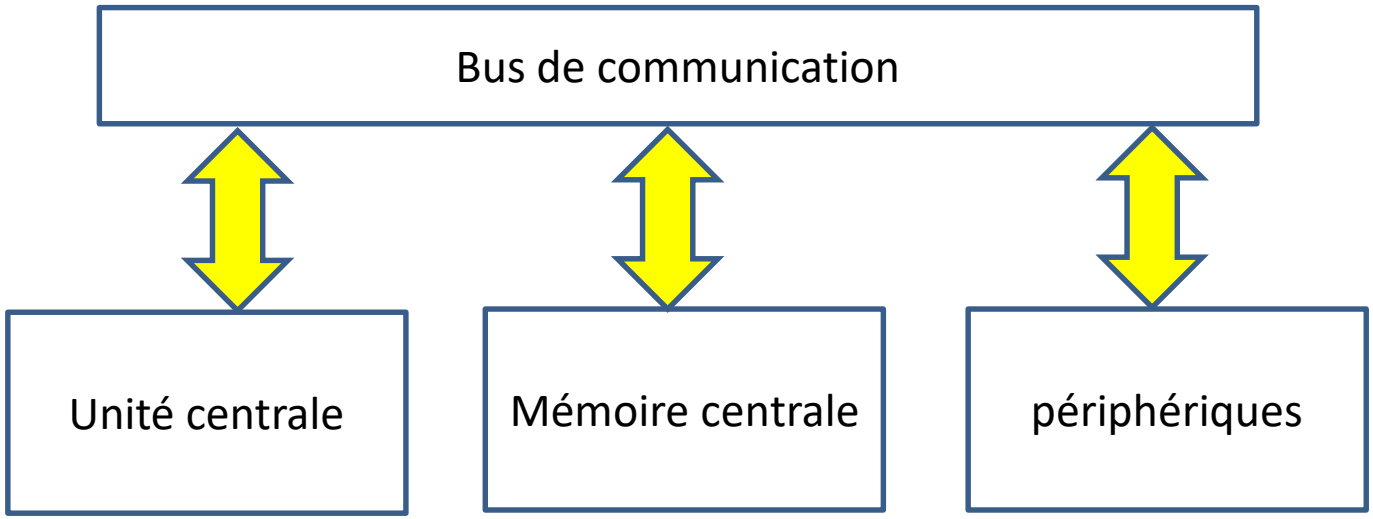
Source : http://en.wikipedia.org/wiki/Transistor_count

Architecture de Von Neuman

Un ordinateur comporte:

- **une unité centrale (UC)**
- **des mémoires (contenant données et programmes)**
- **des périphériques**





Architecture de Von Neuman

La Mémoire Centrale

→**Stockage des** données **et des** programmes
→**codés par des suites de 0 et de 1**
→**Chaque** cellule **mémoire est désignée par son** adresse
→**Toutes les cellules ont la même taille (**mot**), exprimée en nombre de** bits **ou d'**octets**.**

Opérations sur une cellule:

lecture du contenu ***Adresse*** ***Valeur***
écriture d'une ***Valeur*** **à une** ***Adresse***

Architecture de Von Neuman

Les périphériques

Claviers, écrans, souris, crayons optiques, lecteurs de codes à barre, capteurs, synthétiseurs vocaux / musicaux, lecteurs de disquettes, numériseurs (scanners), modems, imprimantes, unités de disques, bandes magnétiques, disques optiques, traceurs, réseaux, tablettes de projection, etc.

Classification possible :

- **périphériques d'entrée**
- **périphériques de sortie**
- **périphériques de stockage**

2.Codage(s) de l'information

- Utilisation

→le stockage en mémoire

→la manipulation des données

→la communication avec les périphériques

- Architecture de Von Neumann

2.Codage(s) de l'information

- Quelques rappels:

$$\underbrace{2\times 2\times 2.....\times 2}_{\text{a fois}} = 2^{a}$$

$$\frac{\overbrace{2\times 2\times 2.....\times 2}^{\text{a fois}}}{\underbrace{2\times 2\times 2.....\times 2}_{\text{b fois}}} = 2^{a-b}$$

a >b dans cet exemple

2.Codage(s) de l'information

$$\underbrace{\frac{2\times 2\times 2.....\times 2}{2\times 2\times 2.....\times 2}}_{\text{a fois}} = 1 = 2^{a-a} = 2^{0}$$

$$\frac{1}{2\times 2\times 2.....\times 2} = 2^{-a}$$

$$\frac{1}{2} = 2^{-1}$$

2.Codage(s) de l'information

- **Un octet : 2^8 = 256 valeurs différentes**

- **<u>Codage des nombres en binaire non-signé</u> : sur un octet, nombres de 0 à 255**

contenu	**1**	**0**	**1**	**0**	**0**	**0**	**1**	**1**
Poids	**128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**
	2^7	**2^6**	**2^5**	**2^4**	**2^3**	**2^2**	**2^1**	**2^0**
total = 163	**128**		**32**				**2**	**1**

contenu	**c_7**	**c_6**	**c_5**	**c_4**	**c_3**	**c_2**	**c_1**	**c_0**
Poids	**2^7**	**2^6**	**2^5**	**2^4**	**2^3**	**2^2**	**2^1**	**2^0**

total = $c_7 . 2^7 + c_6 . 2^6 + c_5 . 2^5 + c_4 . 2^4 + c_3 . 2^3 + c_2 . 2^2 + c_1 . 2^1 + c_0 . 2^0$

2.Codage(s) de l'information

- **Nombre à virgule**

contenu	**1**	**,**	**1**	**1**	**0**	**1**
Poids	**1**		**0.5**	**0.25**	**0.125**	**0.0625**
	2^0		2^{-1}	2^{-2}	2^{-3}	2^{-4}
total = 1.8125	**1**		**0.5**	**0.25**		**0.0625**

- Ces nombres sont codés par « la notation virgule flottante » simple précision, double précision ou précision étendue.
- **Nombre en base n** : est composé de chiffres < n

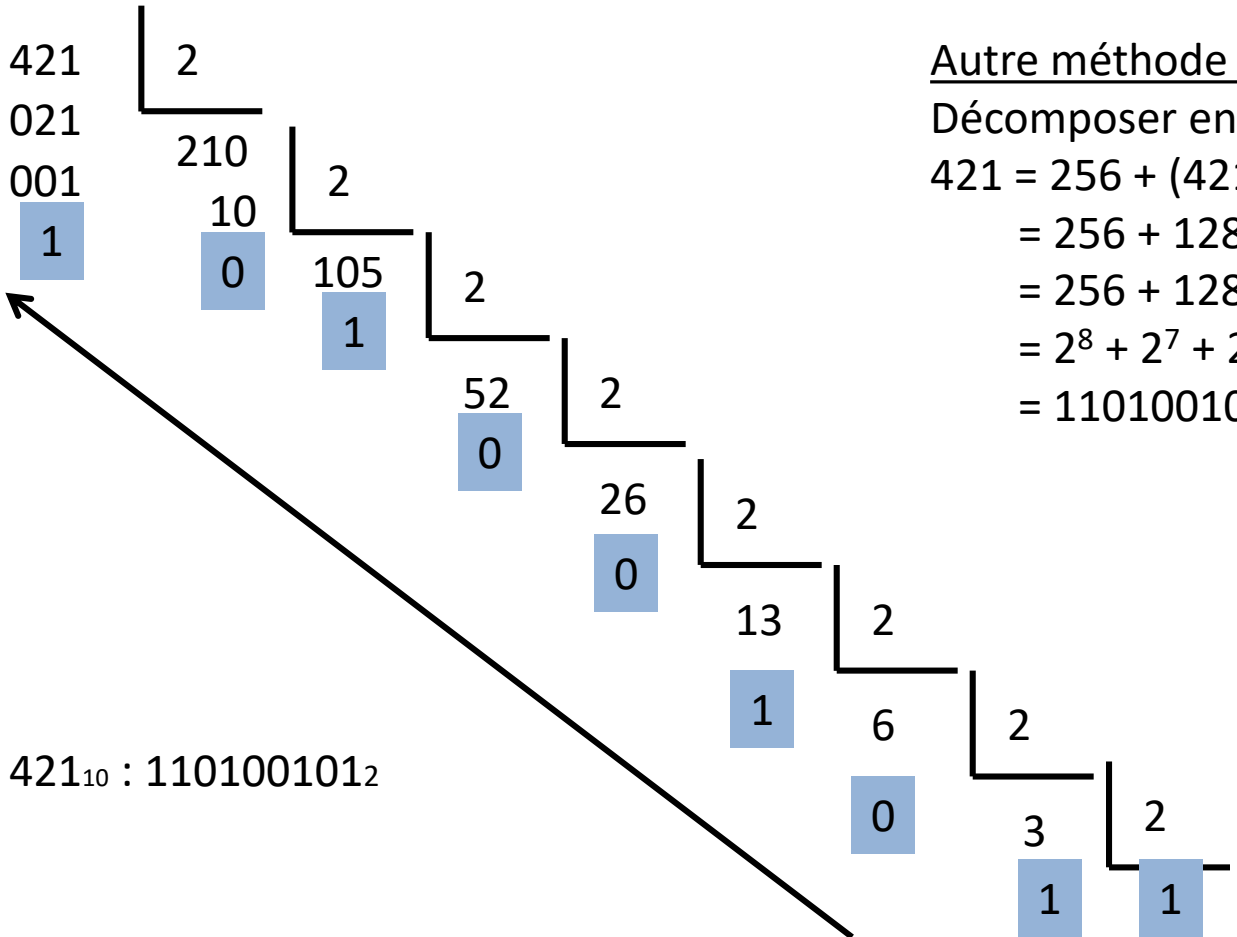
contenu	c_3	c_2	c_1	c_0	,	c_{-1}	c_{-2}	c_{-3}	c_{-4}
Poids	n^3	n^2	n^1	n^0		n^{-1}	n^{-2}	n^{-3}	n^{-4}

total $= c_3 . n^3 + c_2 . n^2 + c_1 . n^1 + c_0 . n^0 + c_{-1} . n^{-1} + c_{-2} . n^{-2} + c_{-3} . n^{-3} + c_{-4} . n^{-4}$

- **Nombre en hexadécimal : base 16 (chiffres < 16 de 0 à 15)**
  - **raccourci de l'écriture d'un quartet en binaire**
  - **0 (0000), 1 (0001), 2 (0010), 3 (0011), 4 (0100), 5 (0101), 6 (0110), 7 (0111), 8 (1000) 9 (1001), A (1010), B (1011), C (1100), D (1101), E (1110) et F (1111)**

2.Codage(s) de l'information

Passage d'un nombre décimal en un nombre en base n
Méthode des divisions successives pour la partie entière
Exemple: convertir 421 en base 2



$421_{10} : 110100101_2$

Autre méthode pour le binaire:
Décomposer en puissance de 2
421 = 256 + (421-256)
= 256 + 128 + (165-128) = 256 + 128 + 37
= 256 + 128 + 32 + 4 + 1
= $2^8 + 2^7 + 2^5 + 2^2 + 2^0$
= 110100101_2

2.Codage(s) de l'information

Passage d'un nombre décimal en un nombre en base n

Méthode des multiplications successives pour la partie décimale

Exemple: convertir 0,67 en base 2

$0,67 = 0,a_0a_1a_2a_3\ldots$ où $a_i = 0$ ou 1

$2 \times 0,67 = 1,34 = a_0,a_1a_2a_3\ldots. \Rightarrow a_0 = 1$

$0,34 = 0,a_1a_2a_3\ldots$

$2 \times 0,34 = 0,68 = a_1,a_2a_3a_4\ldots \Rightarrow a_1 = 0$

$2 \times 0,68 = 1,36 = a2,a3a4\ldots. \Rightarrow a_2 = 1$

Etc…

0,67 = 0,10101011…

0,67
X 2

1,34
X 2

On enlève 1 avant multiplication

0,68
X 2

1,36
X 2

0,72
X 2

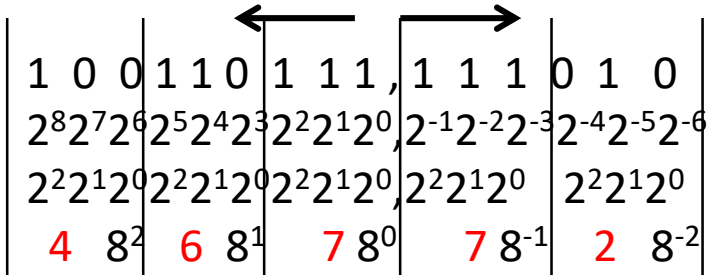
1,44
X 2

0,88

2.Codage(s) de l'information

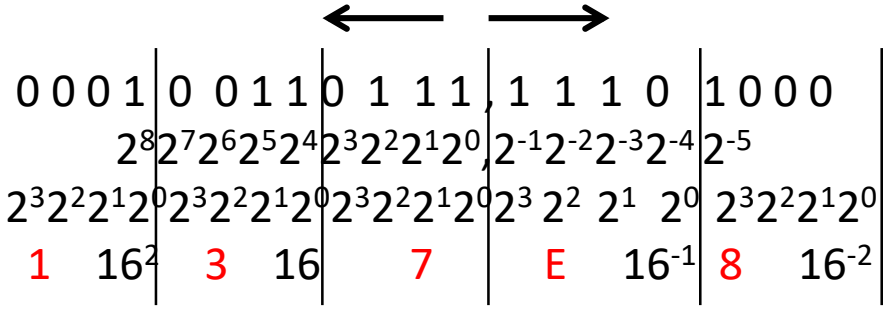
Conversion d'un nombre binaire en octal (base 8) et en hexadécimal (base 16)

Octal
Puissance de 8
$8 = 2^3$ groupe de 3
$2^6 = 8^2$
$2^3=8$



$100110111,11101_2 = 467,72_8$

Hexadécimal
Puissance de 16
$16 = 2^4$ groupe de 4



$100110111,11101_2 = 137,E8_{16}$

2.Codage(s) de l'information

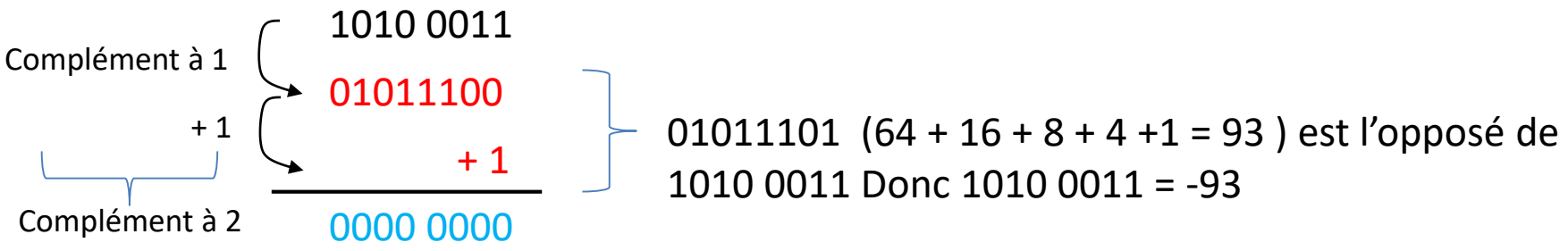
- <u>Codage en complément à deux (excédent 256).</u>

  Sur un **octet**, codage de -128 à +127 :

  1010 0011 163 - 256 = -93

a et b sont opposés si et seulement si a+b=0. Quel est l'opposé de 1010 0011?

(Rappel : 1111 1111 +1 = 1 0000 0000 = 0000 0000 sur 8 bits)



-128	-127	-1	0	1	127	128
1000 0000	1000 0001	1111 1111	0000 0000	0000 0001	0111 1111	0 1000 0000

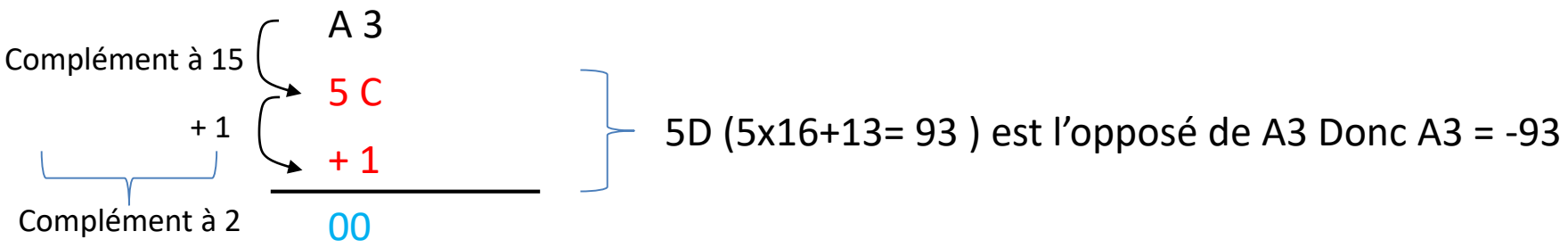
Le bit de poids fort (pF) donne une indication sur le signe: 1 négatif et 0 positif sur un nombre de bits fixé ici 8

2.Codage(s) de l'information

- <u>Codage en complément à deux et notation hexadécimale.</u>

  soit XY un nombre en hexadécimal représentant deux quartets soit un octet

  Le nombre est négatif si le bit de poids fort du quartet de poids fort est à 1 (au moins 1000_2) soit si ce quartet X est supérieur ou égal à 8.



-128	-127	-1	0	1	127
80	81	FF	00	01	7F

Codage(s) des caractères : Code ASCII

Préambule: Pour représenter des caractères dans un fichier texte, on associe une séquence de bits (code) à une lettre, un chiffre ou un symbole.

Le premier codage largement répandu est l'ASCII (American Standard Code for Information Interchange) créé en 1967:

-Code de 7 bits (0 à 127 (2^7)) caractères anglais, nombres de 0 à 9 et certains caractères spéciaux.

- Il ne définit pas les codes de 128 à 255 (pas de lettres accentuées)

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Codage(s) des caractères : ISO (codes ASCII étendus)

ISO (International Organization for Standardisation) a crée des standards pour les codes de 128 à 255, le plus connu :
ISO – 8859 – 1 (Latin 1) inclue les langues européennes

ISO-8859-1																
	x0	**x1**	**x2**	**x3**	**x4**	**x5**	**x6**	**x7**	**x8**	**x9**	**xA**	**xB**	**xC**	**xD**	**xE**	**xF**
0x	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1x	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2x	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6x	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL
8x	PAD	HOP	BPH	NBH	IND	NEL	SSA	ESA	HTS	HTJ	VTS	PLD	PLU	RI	SS2	SS3
9x	DCS	PU1	PU2	STS	CCH	MW	SPA	EPA	SOS	SGCI	SCI	CSI	ST	OSC	PM	APC
Ax	NBSP	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
Bx	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
Cx	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
Dx	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
Ex	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
Fx	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Codage(s) des caractères :
Codage universel : Unicode

Unicode: chaque symbole appelé point de code reçoit un nom officiel (par ex « lettre majuscule latine c cédille ») et un index (U+00C7 codé en hexadécimal)
Les points sont regroupés par plages:
-Le latin de base (correspond à ASCII) de 00 à 7F
- Le supplément latin 1 (lettres accentuées Europe de l'ouest) de 80 à 8F
- l'alphabet phonétique international de 250 à 2AF
- les symboles musicaux byzantins de 1D000 à 1D0FF…

UTF-32
Chaque index est codé sur 32 bits. C cédille : 00 00 00 C7
Inconvénient : 4 octets par caractère alors que pour la plupart des caractères utilisés 8 bits suffisent.

Codage(s) des caractères : Codage universel : Unicode

UTF-8 : un codage à taille variable

Les caractères les plus utilisés de 0 à 127 sont codés sur un octet, les caractères de 128 à1023 sur 2 octets …

Le nombre de 1 en bits de poids fort indique le nombre d'octets utilisés

Représentation	**Signification**
0xxx xxxx	1 octet codant 1 à 7 bits
110x xxxx 10xx xxxx	2 octets codant 8 à 11 bits
1110 xxxx 10xx xxxx 10xx xxxx	3 octets codant 12 à 16 bits
1111 0xxx 10xx xxxx 10xx xxxx 10xx xxxx	4 octets codant 17 à 21 bits

Remarques:

- Un texte écrit en ASCII (US) reste inchangé

-Un texte écrit en ISO-8859-1 est modifié pour les lettres accentuées (en UTF-8 sur 2 octets)

- Certaines opérations, comme chercher le nième caractère nécessitent un parcours séquentiel, car on ne peut pas prédire la taille occupée par les n-1 caractères précédents

2. Codage(s) de l'information: Virgule flottante

L'intervalle des nombres utilisés dans un calcul peut être très grand:
masse d'un électron: 9×10^{-28}g
masse du soleil: 2.10^{33}g
Donc si on veut conserver les 34 chiffres avant la virgule de l'un et les 28 chiffres après la virgule de l'autre il faudrait donc 62 chiffres significatifs et la plupart de ces chiffres seraient des 0.
Il faut donc créer un système de représentation des nombres dans lequel l'intervalle des nombres exprimables soit indépendant du nombre de chiffres avant ou après la virgule

2. Codage(s) de l’information: Virgule flottante

Notation en virgule flottante

Exprimer les nombres à l'aide de la notation scientifique:

$n = f*10^e$

f: mantisse

e: nombre entier positif ou négatif, appelé l'exposant

La version informatique de cette notation est l'expression en virgule flottante.

La dimension de l'intervalle est exprimée par l'exposant et la précision par le nombre de chiffre de la mantisse.

C'est une variante de cette représentation qui est utilisée pour les ordinateurs

2. Codage(s) de l’information: Virgule flottante

Pour des raisons d'efficacité, on utilise les bases 2, 4, 8, ou 16 plutôt que 10.

Normalisation

Si le chiffre de la mantisse situé le plus à gauche est à 0, on décale tous les chiffres d'une position vers la gauche et on décrémente l'exposant de 1.

Une mantisse dont le chiffre le plus à gauche est non nul est dite normalisée.

La représentation d'un nombre est alors unique.

La virgule binaire ou hexadécimale est alors supposée être immédiatement à gauche du bit de poids fort de la mantisse.

2. Codage(s) de l'information: Virgule flottante

0	**100 1001**	**, 1101 1100 0000 0000**

On utilise dans ce mode de représentation la méthode de codage par excédent à 64.
La représentation flottante: IEEE 754
Un comité de l'IEEE (Institute of Electrical and Electronics Engineers, association de professionnels des secteurs de l'électronique) s'est constitué avec pour objectif de définir un standard pour les calculs arithmétiques flottants:
- permettre les échanges entre les différents appareils
- offrir une norme précise au concepteur d'ordinateurs

Résultat le standard IEEE 754:
Trois formats de représentation des nombres flottants :
la simple précision sur 32 bits la double précision sur 64 bits la précision étendue sur 80 bits

2. Codage(s) de l'information: Virgule flottante

Simple précision

Bit de signe	Exposant 8 bits	Mantisse sur 23 bits

Double précision

Bit de signe	Exposant 11 bits	Mantisse sur 52 bits

L'exposant est codé excédent à 127 pour la simple précision et en excédent à 1023 pour la double.
Une mantisse est dite normalisée quand le premier bit qui suit la virgule vaut 1. Par hypothèse du standard IEEE la mantisse est normalisée (standard), donc le premier bit est toujours égal à 1, on peut donc s'en passer et noter implicitement sa présence. Ainsi une mantisse IEEE comprend un bit supposé à 1 qu'on appelle bit caché, puis 23 ou 52 bits de valeur quelconque, de même la virgule est implicite **après** le bit caché. Donc la mantisse est appelée dans le standard IEEE pseudomantisse ou significande.

3. Architecture des microprocesseurs

- Unité Centrale du microprocesseur
- Notion de programmation bas niveau
- Mémoire cache

3.1 Unité centrale du microprocesseur

- **<u>But</u> : mettre en évidence les principaux constituants d'une <u>Unité Centrale</u>**
- **<u>Préambules</u> :**
  - **ALU ou UAL (Unité Arithmétique et Logique) permettant de réaliser différentes *opérations* grâce à un *décodeur***
  - **MEMOIRE (circuit de mémoire) et en particulier les lignes RD/WR associées, CS (chip select), les bus d'adresses et de données : ce circuit mémoire permet de stocker (W) de l'information en vue d'un usage futur (R).**

***Remarques* :**

→l'UAL est un composant de l'UC

→le circuit de mémoire peut être considéré comme un périphérique particulier

3.1 Unité centrale du microprocesseur

- **Problème :**
  exécutions des instructions *évoluées* à l'aide d'instructions *machine* élémentaires

```
C := A + B ;                              {1}
if C > 0 then      action_1 ;             {2}
                   action_2 ...
...
```

- **Comment est représenté chaque identificateur (A, B, C) avant la phase d'exécution ?**
  →par une *zone de mémoire d'adresse connue* que nous noterons &A, &B ou &C (arbitrairement et symboliquement)

3.1 Unité centrale du microprocesseur

```
C := A + B ;                                    {1}
if C > 0 then        action_1 ;                 {2}
                     action_2 ...
...
```

- **Pour coder l'instruction {1} on peut imaginer :**
  - **une seule instruction**

    add &A, &B, &C ; C←[&A]+[&B]

  Or, il faut passer par une zone mémoire tampon

  - **trois instructions différentes utilisant explicitement l'ACCU (zone mémoire interne à l'UC: registre accumulateur) :**

    load &A ; ACCU ← [&A]

    add &B ; ACCU ← [ACCU]+[&B]

    store &C ; C ←[ACCU]

3.1 Unité centrale du microprocesseur

```
C := A + B ;                                  {1}
if C > 0 then      action_1 ;                 {2}
                   action_2 ...
...
```

- **Codage de l'instruction {2} :**

```
21    load &C ; ACCU ← [&C]                                  {2.1}
22    comp 0   ; [ACCU] :: 0                                 {2.2}
23    jg 47    ; aller_a_l'adresse  47 si [ACCU]>0           {2.3}     jg : jump if greater
24    ...
```

{2.1} : charge dans ACCU le contenu de C

{2.2} : compare [ACCU] et 0 et positionne *un* indicateur

(*l'indicateur correspondant est mis à VRAI et les autres à FAUX*) selon que:

[ACCU] >0 : *greater_flag* ← VRAI

[ACCU] <0 : *lower_flag* ← VRAI

[ACCU] =0 : *zero_flag* ← VRAI

{2.3} : si *greater_flag* est VRAI, alors la prochaine instruction exécutée sera l'instruction 47, sinon l'instruction 24.

3.1 Unité centrale du microprocesseur

```
C := A + B ;                                  {1}
if C > 0 then        action_1 ;               {2}
                     action_2 ...
...
```

- **Codage de l'instruction {2} :**

21	**load &C ; ACCU ← [&C]**		**{2.1}**	
22	**comp 0**	**; [ACCU] :: 0**	**{2.2}**	
23	**jg 47**	**; aller_a_l'adresse 47 si [ACCU]>0**	**{2.3}**	jg : *jump if greater*
24	**...**			

Remarques :

Mise en évidence de

***une zone spéciale (registre) contient l'adresse de l'instruction à exécuter par la CPU (UC)**

***elle est contenue dans le *compteur ordinal* ou PC.**

***Une instruction telle que "jg" peut modifier implicitement la valeur de ce registre.**

3.1 Unité centrale du microprocesseur

Etude d'un cas d'école

- *principales caractéristiques :*

  mots de 8 bits pour instructions et données

  adresses sur 6 bits (⇒64 mots adressables)

  un registre ACCU de 8 bits

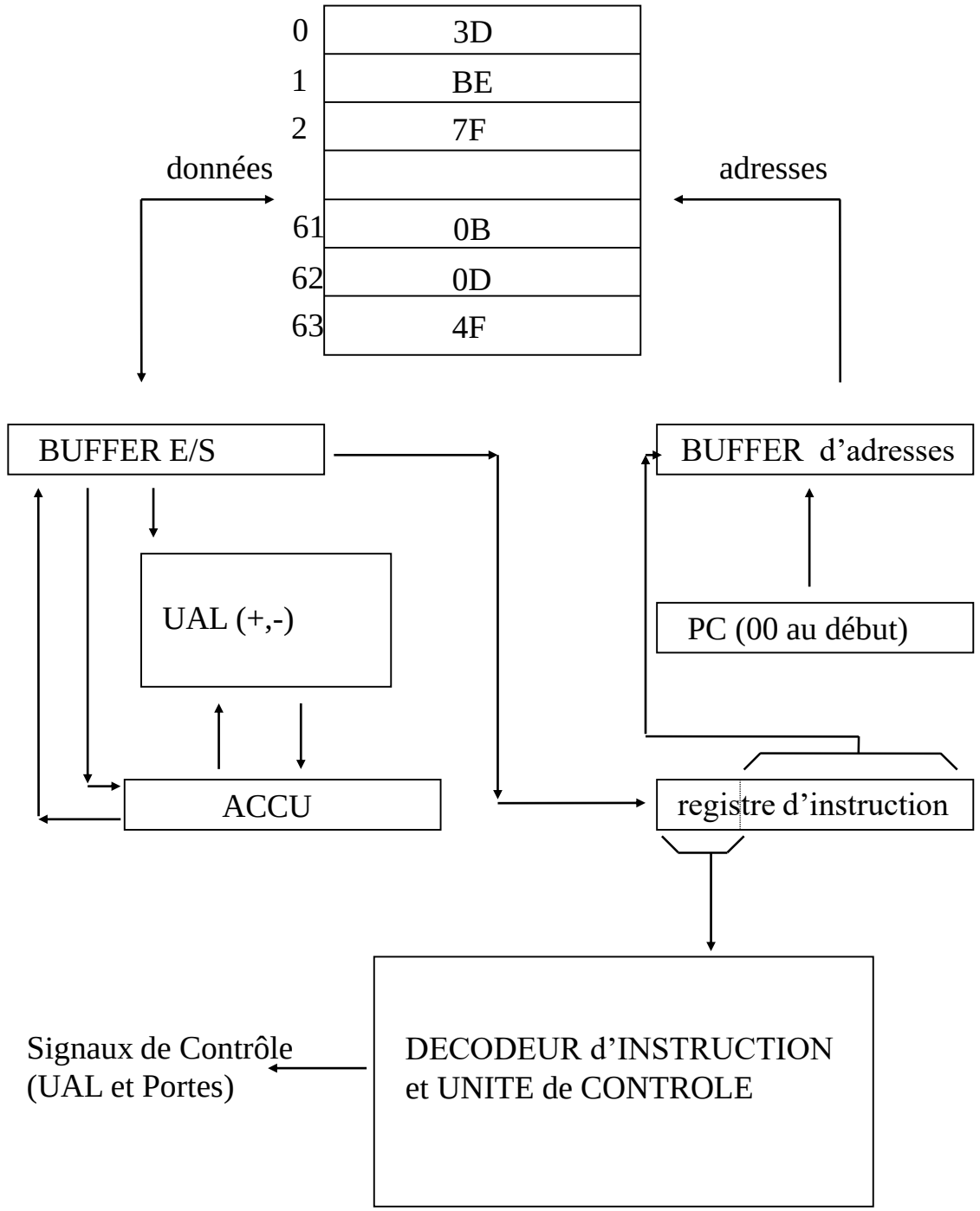
  un jeu de 4 instructions de format général :

8	**7**	**6**	**5**	**4**	**3**	**2**	**1**
Code op (2 bits)		Adresse opérande (6 bits)					

- **Jeu d'instructions :**

forme mnémonique	**(code)**	**signification**
load M	**(00)**	**ACCU←[M]**
store M	**(01)**	**M←[ACCU]**
add M	**(10)**	**ACCU←[ACCU]+[M]**
sub M	**(11)**	**ACCU←[ACCU]-[M]**

la suite des opérations ...?



3.1 Unité centrale du microprocesseur

Etude d'un cas d'école

Remarques :

- **le cycle d'exécution d'une instruction se divise en deux phases :**
  - **"fetch" ou recherche d'instruction (et chargement dans le registre d'instruction)**
  - **exécution proprement dite**

- **Modes de fonctionnement via des données**
  - **ECRITURE : adresse dans buffer adresse**
    - ✓ **donnée dans buffer d'E/S → mémoire**
    - ✓ **valider bascule ECRITURE**
  - **LECTURE : adresse dans buffer adresse**
    - ✓ **valider bascule LECTURE**
    - ✓ **donnée dans mémoire → buffer d'E/S**

Note : un buffer est un registre lié à un bus

3.2 Notions de programmation bas niveau

classification générale des instructions :

- **C1 : mouvements de données entre registres et mémoire centrale**

  mov ax, mot ; *ax ←[MOT] (16 bits)*
- **C2 : instructions arithmétiques et logiques : +, -, *, /, in(dé)crémentation , ET, OU, décalage, rotation, comparaison, complémentation, ...**

  add bl, cl ; *bl ← [bl] + [cl] (8bits)*
- **C3 : instructions de contrôle ou branchement conditionnel ou inconditionnel ...**

  jmp LABAS ; *PC ← adresse(LABAS)*
- **C4 : instructions de gestion de l'environnement : mise à jour de la pile, opérations d'E/S, positionnement / lecture des masques de bits d'interruption et indicateurs (bits d'état) ...**

  out PORT, al ; *PORT(d'E/S) ← [al]*

3.2 Notions de programmation bas niveau

Les opérandes:

- ***Exemple* : pour effectuer une addition, on a besoin de deux opérandes au moins (opérateur binaire) et il faut aussi prévoir où mettre le résultat ...**

- **Le *nombre des opérandes* peut être variable :**

  →hlt (halt) est une instruction sans opérande

  →jmp adr_symb : 1 opérande

  →mov reg1, reg2 : 2 opérandes etc...

- **Un *opérande* est localisé à une adresse précise (explicite ou implicite) et on appelle *mode d'adressage* une méthode permettant de calculer l'adresse effective d'un opérande**

3.2 Notions de programmation bas niveau

- **Trois *modes d'adressage* de base :**

- **M1 : adressage direct : jmp AILLEURS**
- **M2 : adressage indirect (lié parfois à l'utilisation de [et]) :**
  mov ax, [bx]
- **M3 : adressage indexé (combiné ou non à l'adressage indirect) : mov ax, [bx][si]**

3.2 Notions de programmation bas niveau

- **On trouve d'autres modes d'adressage dérivés des précédents :**
- **adressage immédiat :**

  mov al, 5 ; al ← 5
- **adressage par *pile* (opérande implicite) :**

  concerne les opérations,

  push ; pousser un mot dans la pile (sommet)

  pop ; sortir un mot de la pile (sommet)

EXERCICE : que fait le petit programme assembleur suivant ?

```
                .
        xor     ax, ax
        xor     si, si
        lea     bx, TABLEAU
        mov     cx, 10
BOUCLE: add     ax, [bx][si]
        inc     si
        inc     si
        loop    BOUCLE
                .
                .
                .
```

indication **: loop BOUCLE**

cx ←cx-1

saut à BOUCLE si cx!=0

sinon en séquence ...

Remarque **: registre "si" et instruction "loop" à partir du 8086/8088 seulement**

3.2 Notions de programmation bas niveau

Réalisation d'une instruction

Deux possibilités pour implémenter les instructions interprétées par l'UC :

1. **"câblage" : instruction → fonction booléenne → circuit (technologie adaptée)**
   - **avantage : rapide à l'exécution**
   - **inconvénient : complexe, cher et figé**
2. **"microprogrammation" : une instruction est considérée comme une fonction ou une procédure constituée de micro-instructions câblées ; l'instruction est enregistrée en mémoire morte**
   - **avantage : circuit du processeur simple et jeu d'instructions modifiables**
   - **inconvénient : vitesse d'exécution assez peu élevée**

3.2 Notions de programmation bas niveau

Réalisation d'une instruction

Lorsque le jeu d'instructions est limité, on peut procéder de la manière suivante :

→ *émulation* ou simulation logicielle (instruction → fonction assembleur ou langage C) ; inconvénient : "trés" lent ...
→ajout d'un *co-processeur* câblé ou microprogrammé (processeur spécialisé qui exécute des instructions spécifiques) ; l'Unité Centrale sous-traite ...

***Exemple* : le co-processeur arithmétique 80xx7 associé au 80xx6 réalise les opérations en Virgule Flottante et les fonctions mathématiques**

Principes généraux de conception d'une unité centrale

Constitution d'une Unité Centrale

- **le processeur est caractérisé par le nombre très élevé de signaux à transmettre/recevoir**

  → problème lors de la conception /réalisation d'un µP
- **Le µP est réalisé**

  → sur une plaquette de silicium (5mmx15mm) ;

  →enchâssé dans un boîtier plastique entouré de broches →le contact avec l'extérieur
- **Insuffisant du nombre de broches**

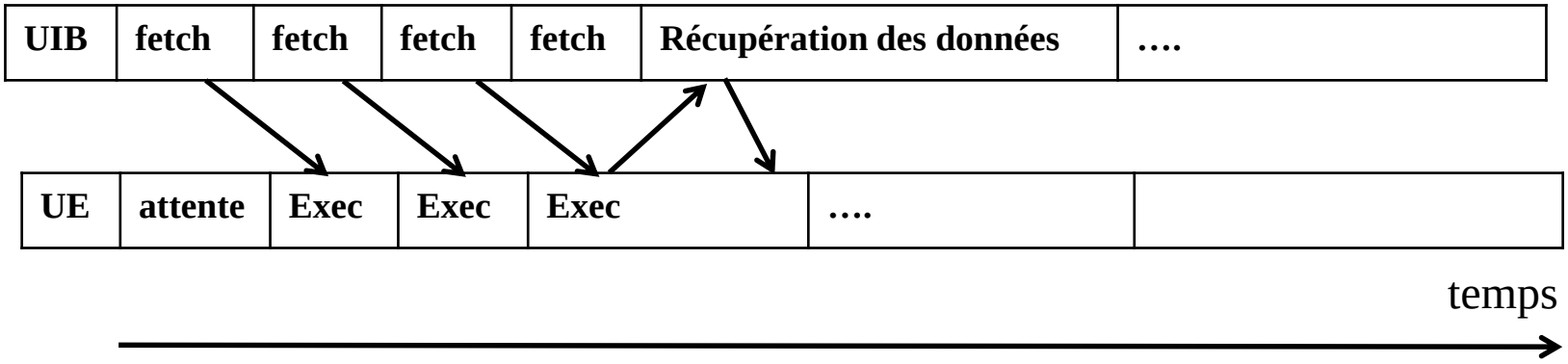
  ⇒ certaines broches combinent *plusieurs fonctions* : nécessité de multiplexage
- **L'horloge (interne ou externe au µP, commandée par un quartz) est indispensable pour cadencer le fonctionnement du µP**
- **Le jeu d'instructions de base peut être étendu grâce au *co-processeur***

Principes généraux de conception d'une unité centrale

Les µprocesseur CISC d'Intel → description générale du 8088 d'Intel

Architecture « pipeline »

→ Pour remédier à ce temps d'attente, le *prétraitement* ou *traitement pipeline* a été introduit dans le 8086/8088. Pendant que l'UE exécute les informations qui lui sont transmises, l'instruction suivante est chargée dans l'UIB. Les instructions qui suivront sont placées dans une file d'attente. Lorsque l'UE a fini de traiter une instruction l'UIB lui transmet instantanément l'instruction suivante, et charge la troisième instruction en vue de la transmettre à l'UE. De cette façon, l'UE est continuellement en activité.



3.3 Mémoire cache

- la lecture et l'écriture d'informations
- Ce sont des RAM (Random Access Memory) mémoire dont le temps d'accès à l'information est le même quelque soit le mot sollicité ⇒ mémoires vives

Deux familles de RAM: statiques et dynamiques

•Les *RAM statiques* (étudiées en cours) semblables aux **bascules D**
⇒garantir la mémorisation de l'information aussi longtemps que l'alimentation électrique est maintenue sur la mémoire

•Les *RAM dynamiques* sont composées d'un ensemble de petits **condensateurs**, chacun pouvant recevoir ou restituer une charge électrique emmagasinée.
Inconvénient: la charge électrique mémorisée diminue avec le temps
⇒ les rafraîchir une fois toutes les quelques millisecondes → disposer d'un dispositif interne auto rafraîchissement: *les mémoires quasi-statiques*.

3.3 Mémoire cache

❖**Les unités centrales deviennent beaucoup plus rapides que les mémoires principales. Ainsi, lorsque l'unité centrale sollicite la mémoire, elle passe une bonne partie de son temps à attendre que la mémoire réagisse.**

❖**Le microprocesseur doit donc "attendre" la mémoire vive à chaque accès, on dit que l'on insère des "Wait State" dans le cycle d'horloge d'un micro.**

***Ex*: un 386 DX cadencé à 33 MHz ne peut fonctionner sans état d'attente que si les RAM ont un temps d'accès de 40 ns.**

❖**le problème** **technologique : NON**
économique : Oui

*Il est possible de construire des mémoires aussi rapides que les unités centrales mais leur coût, pour des capacités de plusieurs mégaoctets serait prohibitif***.**

❖**Les choix :** *à l'exception des superordinateurs (performance>coût)*
→ **disposer d'une faible quantité de mémoire rapide associée à une quantité importante de mémoire relativement plus lente. Cette mémoire plus rapide est appelée mémoire cache, cache ou antémémoire.**

3.3 Mémoire cache

❖**mémoire cache → mémoire vive statique**

15 ns à 20 ns de temps d'accès

s'insère entre le processeur et la RAM dynamique.

❖**Un contrôleur de mémoire cache est chargé de recopier les instructions et les données les plus fréquemment utilisées par le processeur dans le cache.**

❖**Le principe du cache repose sur deux constatations:**

- en cours d'exécution d'un programme, lorsque le microprocesseur va chercher une instruction en mémoire il y a statistiquement de fortes chances pour que celle-ci se trouve à proximité de l'instruction précédente.

- de plus, les programmes contiennent un grand nombre de structures répétitives de sorte qu'ils utilisent souvent les mêmes adresses.

❖**Gestion de la mémoire cache**

le contrôleur du cache intercepte les adresses émises par le microprocesseur et dans un premier temps recopie le contenu d'un bloc entier de mémoire dans le cache de sorte qu'il y ait une forte probabilité que la mémoire cache contienne les prochaines instructions.

Ce principe permet au microprocesseur, via le contrôleur de cache, d'avoir 90% de chance d'obtenir l'information dans la mémoire cache donc sans "Wait State".

3.3 Mémoire cache

Principe de fonctionnement

❖Le dispositif est constitué de deux éléments principaux:

-la mémoire cache, constituée de RAM
-le contrôleur de mémoire cache comprenant

- **la gestion du cache**
- **un index d'adresses stockant les adresses des blocs contenus dans le cache**
- **un comparateur qui met en correspondance les adresses émises par le microprocesseur et celles contenues dans l'index quand il y a correspondance**

❖l'information est prise dans le cache sinon elle est transférée de la RAM et le contrôleur recopie tout un bloc dans le cache.

3.3 Mémoire cache

Différentes étapes du fonctionnement d'un cache:

1. Le microprocesseur demande une information (instruction ou donnée) I1 située à l'adresse A1 de la mémoire vive
2. Le contrôleur de mémoire cache intercepte la demande et examine sa table d'index pour vérifier si A1 y est présente, et donc si une copie de l'information se trouve dans le cache.
3. Si c'est le cas, l'information est délivrée au microprocesseur depuis la mémoire cache sans temps d'attente
4. Dans le cas contraire, le contrôleur de cache accède à l'information de A1 de la RAM, la délivre à l'unité centrale avec plusieurs temps d'attente et simultanément la recopie en mémoire cache en même temps qu'un bloc d'informations contiguës, parmi lesquelles I2, I3, I4 etc... et actualise sa table d'index.
5. Le microprocesseur réclame l'information suivante, il y a statistiquement 90% de chance pour que ce soit I2, donc présente dans le cache et délivrée dans l'UC sans temps d'attente.

Ce chapitre « Mémoire cache » est extrait de **Architecture de l'ordinateur**
de Andrew Tanenbaum 1 janvier 1996